CS 4713: Introduction to the Internet

Assignment 3: Congestion Control over Cellular Networks

20100006 Mohammad Raza Khawaja

20100029 Syed Mustafa Ali Abbasi

Part 1:

The value of the fixed window size was adjusted in controller.cc, from a minimum of 8 to a maximum of 60. For each test run, the power obtained was recorded. The following graph summarizes the results:

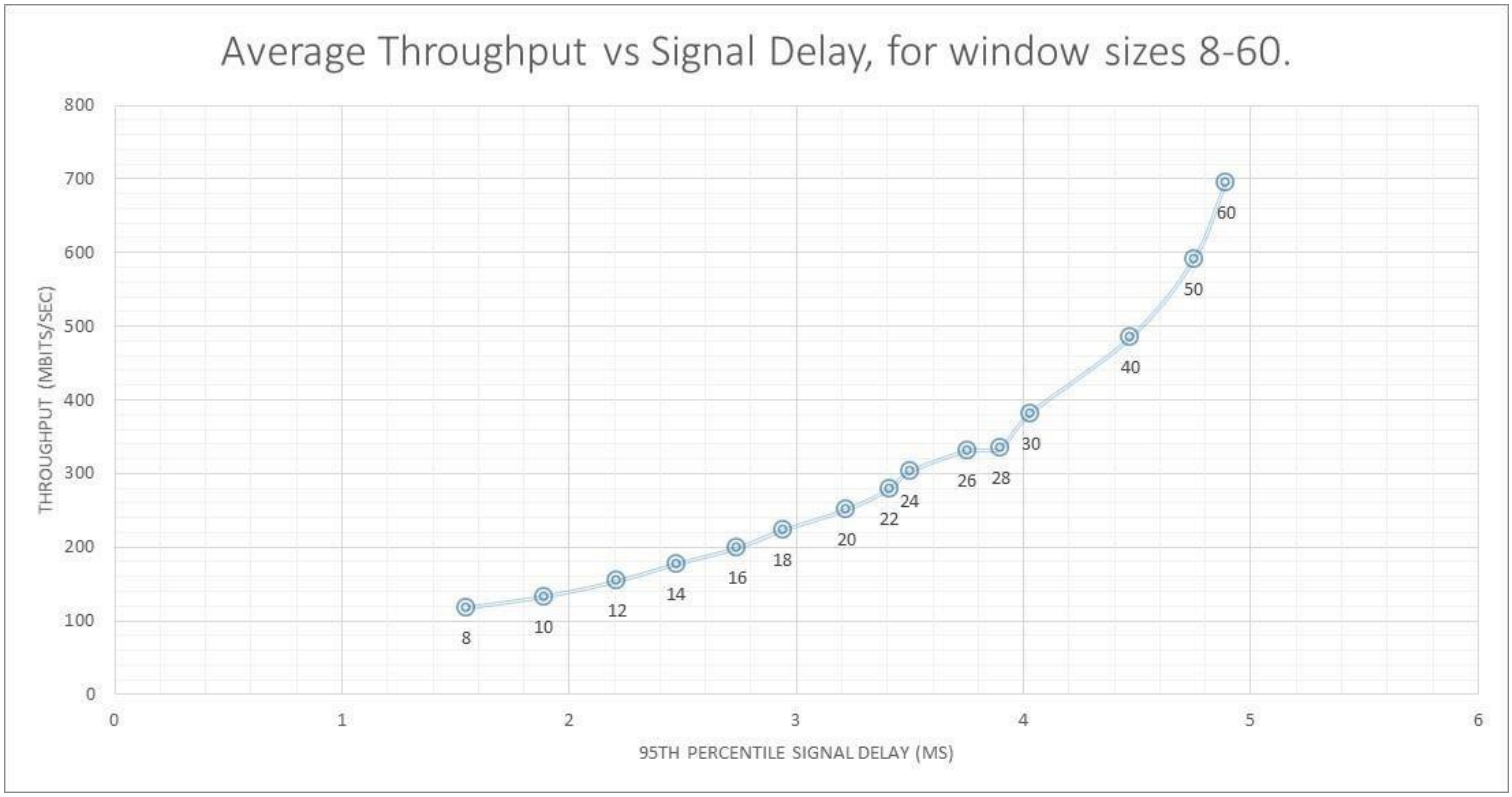


Figure 1.1

As can be observed from Figure 1.1, there is a direct relation between the increasing window size and the delay and throughput that it causes. A smaller window size causes lesser throughput since at any given moment, lesser packets are in flight on the link. And it follows that with lesser amount of packets on the connection at a point, the rate at which the network would be congested would also remain low. As the window size is increased, more packets are sent over the network per unit time, however, delays increase because of congestion, since at this time, no congestion avoidance algorithm was in place.

Figure 1.2.

Figure 1.2 is a graph of the window size against the power (throughput/delay) obtained. The findings obtained here are useful later on when designing congestion control algorithms. As can be seen from the graph, power was maximized when the window size was 12. Sizes below 12 caused the power to decrease, and sizes above 12 caused a decrease in the power as well. This means that a there exists a particular window size that optimizes the ratio between delay and throughput, and the goal of our congestion avoidance algorithm was to have an adaptive window size that was optimized for all any point in time, during the course of transmission.

The following table displays the repeatability of the tests, on the window size for which power was maximized. All values remain fairly constant throughout the five tests, so it can be concluded that the tests are repeatable.

Test Number	Power (Mbits)	Throughput (Mbits/s)	Signal Delay (ms)
1	11.29	2.10	186
2	11.59	2.11	182
3	11.33	2.13	188
4	11.76	2.13	181
5	11.76	2.14	182

Figure 1.3 Checking repeatability of tests on a fixed window size of 12.

Part 2:

Simple AIMD Scheme #1:

The results are for the following algorithm, with a constant timeout set at 100ms, and varying window size.

- On timeout:
  1. ssthresh = ¾ * windowsize
  2. windowsize_new = ssthresh / 2
  3. Enter slowstart phase.
- In SlowStart:
  1. Windowsize += 2 (for each ack received)
- When windowsize >= ssthresh
  1. Exit slowstart.
  2. Start additive increase (windowsize += 1.5/windowsize, for each Ack)

Power	Throughput (Mbits/s)	Signal Delay (ms)	Timeout (ms)	Initial Window Size
27.76	2.97	107	100	25
28.07	2.92	104	100	30
28.19	2.96	105	100	35
27.50	2.97	108	100	40

Since the maximum power was achieved at a window size of 35, the following table shows results for keeping the window size fixed at 35, but varying the timeout – on the same algorithm.

Power	Throughput (Mbits/s)	Signal Delay (ms)	Timeout (ms)	Initial Window Size
26.8	2.65	99	90	35

This configuration made it worse. So to try strike a balance, values of both timeout and window size are varied slightly.

Power	Throughput (Mbits/s)	Signal Delay (ms)	Timeout (ms)	Initial Window Size
26.03	2.63	101	90	30
26.89	2.77	103	95	25

So this means that changing the value of the timeout threshold to anything less than 100 causes lots of packet timeouts, while having too large an initial window size causes congestion. Setting timeout more than 100 would just increase unnecessary delays on each ack.

Simple AIMD Scheme #2:

This scheme was built on the previous, except that a count was kept of the number of timeouts that occurred. If an isolated timeout occurred, then it implied that such drastic steps need not be taken to reduce the window size, but if lots of successive timeouts occurred, then that implies that the window size and the ssthresh need to be adjusted significantly to allow the network to recover.

In this case, if there the number of successive timeouts went greater than 5, then:

- Ssthresh is decreased to window size / 2.
- Window size is decreased to 1.
- Slow start is entered.

On less than 5 successive timeouts, window size is decreased to half of the sstresh, and ssthresh is decreased to 75% of the window size.

The rest of the scheme is AIMD + Slow start, as in scheme #1.

Power	Throughput (Mbits/s)	Signal Delay (ms)	Timeout (ms)	Intital Window Size
27.9	2.99	107	100	35
26.89	2.77	103	95	25

It can be observed that not a huge difference is noted, in fact, the power score actually decreased slightly. Perhaps this scheme is not best suited for this trace, but would work better on a different trace.

Part 3: The Contest

Additive Increase, Additive Decrease (AIAD):

We tried an AIAD approach first and obtained the following results:

Additive Increase	Additive Decrease	Throughput	Signal Delay (ms)	Power
+2.0/cwnd	-15.0/cwnd	3.93	108	36.39
+2.0/cwnd	-12.5/cwnd	4.10	125	32.80
+2.0/cwnd	-17.5/cwnd	3.92	116	33.79
+1.5/cwnd	-15.0/cwnd	3.78	113	33.45

While this approach did give us high values of power (maximum at +2.0, -15.0), we decided to **not** use it since it required hardcoding of several constants and so would not generalize to other network traces.

Slow-Start + Equation Based Algorithm:

We employed an approach similar to the SPROUT-EWMA algorithm. Time was divided into slots of 30 ms each. An Exponential Weighted Moving Average (EWMA) was used to estimate the throughput in each of these time slots. The observed throughput was estimated as:

```
Observed_throughput = number of ACKS in time slot / slot length.
```

Then, the following code was used to update the estimate of the throughout

```
estimated_throughput = (1-alpha)*estimated_throughput +
(alpha)*observed_throughput where alpha = 0.25
```

Since overestimating the throughput is costly in terms of the delay incurred, we also calculated the deviation in the throughput as follows:

```
Dev_throughput = (1-rho)*dev_throughput +
(rho)*|estimated_throughput – observed_throughput|
```

Finally, we used the following equation to calculate and set the window size:

```
CWND = (estimated_throughput – dev_factor * dev_throughput) *
desired_delay
```

where the constants dev_factor and desired_delay were tuned to get a higher power value. The best values turned out to be dev_factor = 0.20 and desired_delay = 60ms. Note that we decrease the estimated throughput by a factor of the deviation in the throughput to ensure that our algorithm does not overestimate the throughput and hence does not exceed the network capacity which can be disastrous.

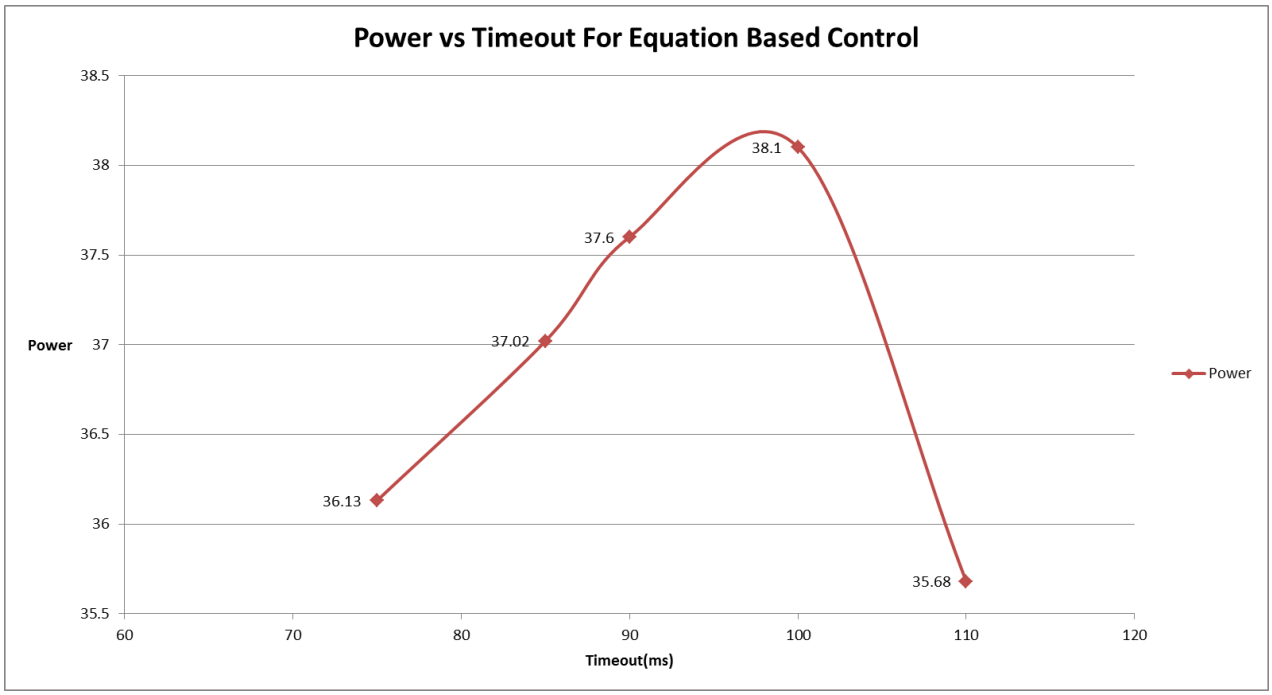
One problem we observed with the above approach was that the window size was initially very small and took some time to start increasing. Therefore, we used a higher initial window size and also incorporated a slow start phase in our algorithm in case we observed very high RTTs (>= 2 * timeout).

We varied the value of fixed timeout value to observe which value gave us the highest value of value. The results are summarized below:

Initial Window Size = 12.0

Multiplicative Decrease Factor in Slow Start = 0.65

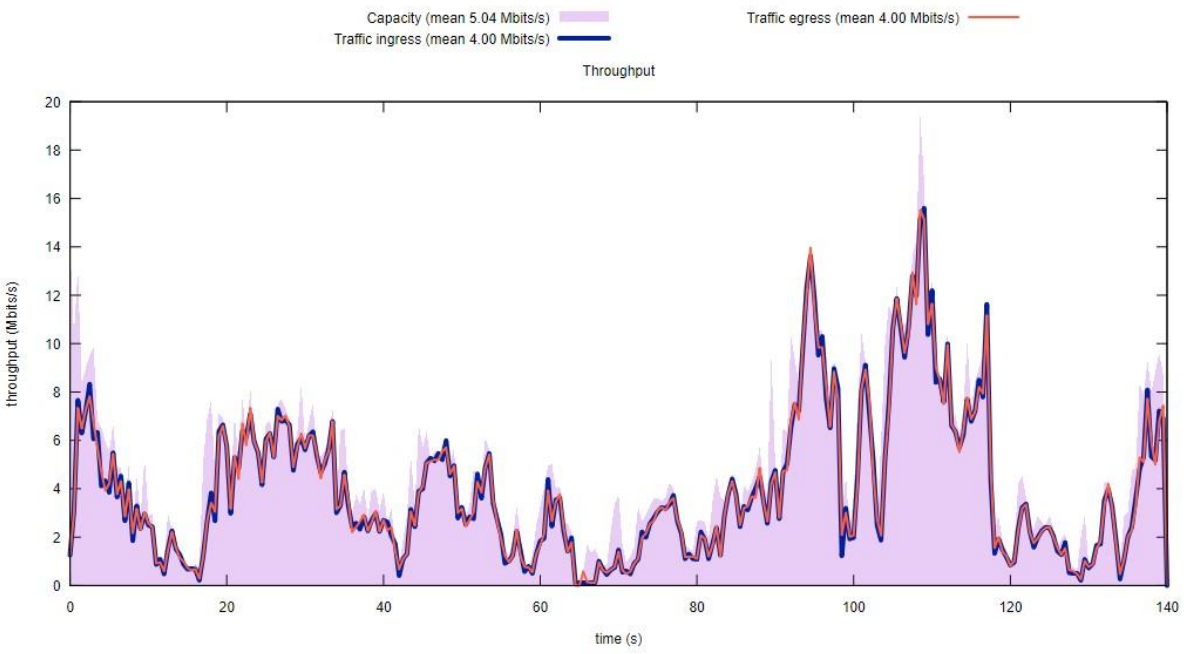
Timeout (ms)	Throughput (Mbits/s)	Signal Delay (ms)	Power
75	3.83	106	36.13
85	3.85	104	37.02
90	3.91	104	37.60
100	**4.00**	**105**	**38.10**
110	3.96	111	35.68



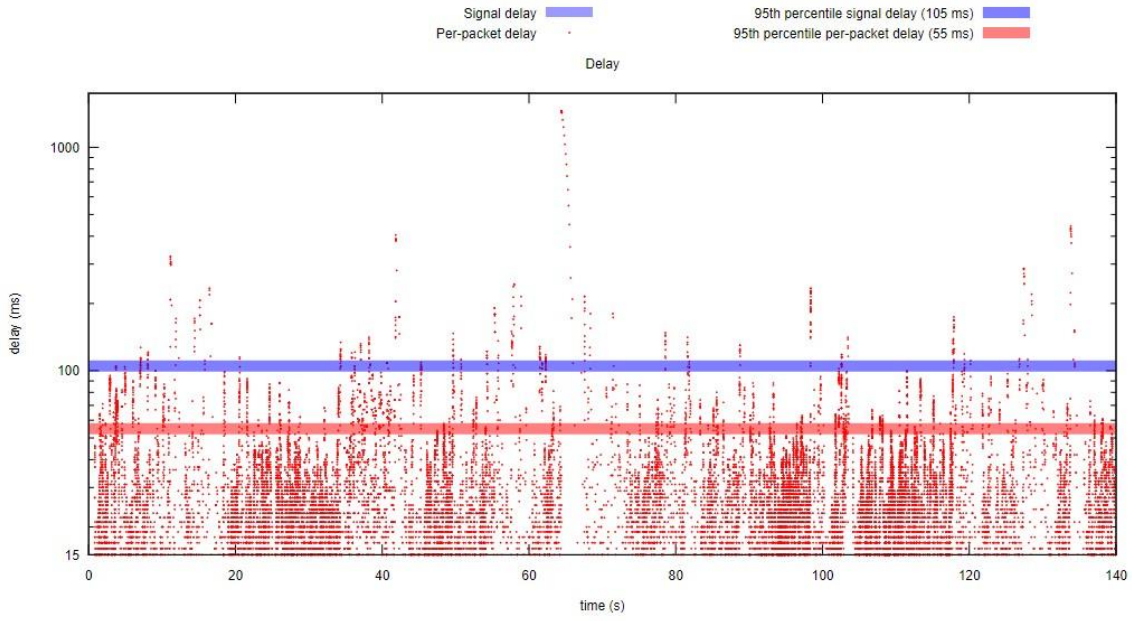
As can be observed from the table and the graph above, the timeout value of 100ms gives us the best power. A lower value lowers the throughput more drastically than lowering the delay whereas a higher timeout value drastically increases the signal delay.

The following graphs show the trend of Throughput vs Time and Delay vs Time obtained for the approach which yielded a power of 38.1.

(Next page follows)



Throughput vs Time



Delay vs Time

We believe that this approach gives us the best results because it aims to maximize throughput in intervals of low delay. It is generally hard to predict when the throughput/network capacity is going to drop suddenly and rapidly and hence difficult to avoid those delays. Hence, this approach tries to keep up with the network capacity (without exceeding it) in times of low delay and lowers the window size when the capacity drops suddenly to ensure no further losses/delays are incurred.